Information Retrieval Techniques for Question Answering

Rakhshani Amin
SN:692

Supervisor
Tsoumakas Grigorias, Assistant Professor

June 4, 2020
Abstract

Throughout History our ability to imprint Information and to access it, made science progress. First Information was made accessible through books and libraries, till Internet made an appearance and knowledge was made more accessible that ever. Now we are facing another big problem that if we let it grow it might even equalize the lack of knowledge. Ely et al. [1] have found that physicians spend an average of 2 min or less in seeking an answer, while Hersh et al. [2] have found that it takes more than 30 min on average for a health care professional to search for an answer. As a result many clinical questions go unanswered. “Better be ignorant of a matter than half know it.” Publilius Syrus said. So we know as a fact that efficient searching has gone far from the reach of plain human abilities. Thus we need Question Answering (QA) Systems, which eventually will do the hard work of answering in a matter of few minutes. Question Answering is a multi-layered problem. In this work we will try to analyze the potential of re-ranking results retrieved from Information Retrieval techniques. More specifically the re-ranking is performed by machine learning models using word embedding that try to take into account the context of the sentences.
# Contents

<table>
<thead>
<tr>
<th>Abstract</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Introduction</td>
</tr>
<tr>
<td>1.1 Description</td>
</tr>
<tr>
<td>1.2 Purpose</td>
</tr>
<tr>
<td>1.3 Structure</td>
</tr>
<tr>
<td>2 Background</td>
</tr>
<tr>
<td>2.1 Machine Learning</td>
</tr>
<tr>
<td>2.2 Information Retrieval</td>
</tr>
<tr>
<td>2.3 Natural Language Processing</td>
</tr>
<tr>
<td>2.4 Metrics</td>
</tr>
<tr>
<td>2.5 Context</td>
</tr>
<tr>
<td>2.6 Evolution of Word Embedding</td>
</tr>
<tr>
<td>2.6.1 Bag of Words (BoW)</td>
</tr>
<tr>
<td>2.6.2 Distributional Embeddings</td>
</tr>
<tr>
<td>2.6.3 Word Embeddings</td>
</tr>
<tr>
<td>2.6.4 Neural Networks for NLP</td>
</tr>
<tr>
<td>2.6.5 Sequence to sequence models</td>
</tr>
<tr>
<td>2.6.6 Attention</td>
</tr>
<tr>
<td>2.6.7 Memory Based Networks</td>
</tr>
<tr>
<td>2.6.8 Pretrained Language Models</td>
</tr>
<tr>
<td>3 Experimental work</td>
</tr>
<tr>
<td>3.1 Overview</td>
</tr>
<tr>
<td>3.2 Data Description</td>
</tr>
<tr>
<td>3.2.1 Pubmed Dataset</td>
</tr>
<tr>
<td>3.2.2 BioASQ Challenge</td>
</tr>
<tr>
<td>3.3 Bioasq Results</td>
</tr>
<tr>
<td>3.4 Elasticsearch Analysis</td>
</tr>
<tr>
<td>3.5 Query and Filter context</td>
</tr>
<tr>
<td>3.6 Full Text Queries</td>
</tr>
<tr>
<td>3.7 Multi Match Query</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Description

PubMed Central® (PMC) is a free full-text archive of biomedical and life sciences journal literature at the U.S. National Institutes of Health’s National Library of Medicine (NIH/NLM). In keeping with NLM’s legislative mandate to collect and preserve the biomedical literature, PMC serves as a digital counterpart to NLM’s extensive print journal collection. Every year many articles are being published and are added to the PubMed data pool.

Thus rises the need to be able to handle all these data effectively. Who would be better than a person that in some way learns all that knowledge, perceives all meaning and relates knowledge across all publications. Hence that person will be in the position of giving answers to any knowledge included in that material effectively. We intuitively know that we people are not so good in the described task. That is the field of Question Answering and it’s goal is the described task. Not only to find related documents but also understand context and be able to give proper answers that may be included in more than one source. This would not only help us have answers for which we should search extensively but also find probable ambiguities in some theories in the scientific community and thus help scientific research become more effective. Without repetitions or ambiguities.

In this work we focus on the first part, which is finding the proper sources through state of the art methodologies that are said to perceive context. In the general case one can point out 12 key objectives in the field of the QA-systems building developed in 2002 by a group of researchers [3]. The first step is question classification, that is to say, determining in which scientific or general category the question is about. In this particular work we will skip this step by taking as granted that we are working on the Biomedical domain. All questions and answers are related to Biomedicine.
1.2 Purpose

The purpose of this work is to study and implement the first part of a QA System as described above. To discover the state of the art procedures that exists for context ranking and see how this elevates us from the traditional information retrieval procedures. In this way we will be able to also compare between the different Machine Learning Techniques in question.

1.3 Structure

In the second chapter we discuss about basic concepts and technologies that are being used in QA systems and we have a discussion about the evaluation methodologies and metrics that will be used. In the third chapter we introduce the new state of the art methodologies that are being used for context comparison in text. These are being developed by huge companies and universities like Google, Facebook, Stanford University etc. In the fourth chapter we introduce the way this work was developed and the techniques it is using in order to reach its goals. In the last but not least chapter we put forward any conclusions that were drawn from this work and we introduce future work and possible research.
Chapter 2

Background

2.1 Machine Learning

Machine learning is an application of artificial intelligence (AI) that provides systems the ability to automatically learn and improve from experience without being explicitly programmed. Machine learning focuses on the development of computer programs that can access data and use it to learn for themselves.

The process of learning begins with observations or data, such as examples, direct experience, or instruction, in order to look for patterns in data and make better decisions in the future based on the examples that we provide. The primary aim is to allow the computers learn automatically without human intervention or assistance and adjust actions accordingly.

Some machine learning methods

- **Supervised machine learning algorithms** can apply what has been learned in the past to new data using labeled examples to predict future events. Starting from the analysis of a known training dataset, the learning algorithm produces an inferred function to make predictions about the output values. The system is able to provide targets for any new input after sufficient training. The learning algorithm can also compare its output with the correct, intended output and find errors in order to modify the model accordingly.

- In contrast, **unsupervised machine learning algorithms** are used when the information used to train is neither classified nor labeled. Unsupervised learning studies how systems can infer a function to describe a hidden structure from unlabeled data. The system doesn’t figure out the right output, but it explores the data and can draw inferences from datasets to describe hidden structures from unlabeled data.

- **Semi-supervised machine learning algorithms** fall somewhere in between supervised and unsupervised learning, since they use both labeled and unlabeled data for training – typically a small amount of labeled data and a
large amount of unlabeled data. The systems that use this method are able to considerably improve learning accuracy. Usually, semi-supervised learning is chosen when the acquired labeled data requires skilled and relevant resources in order to train it / learn from it. Otherwise, acquiring unlabeled data generally doesn’t require additional resources.

- **Reinforcement machine learning algorithms** is a learning method that interacts with its environment by producing actions and discovers errors or rewards. Trial and error search and delayed reward are the most relevant characteristics of reinforcement learning. This method allows machines and software agents to automatically determine the ideal behavior within a specific context in order to maximize its performance. Simple reward feedback is required for the agent to learn which action is best; this is known as the reinforcement signal.

Machine learning enables analysis of massive quantities of data. While it generally delivers faster, more accurate results in order to identify profitable opportunities or dangerous risks, it may also require additional time and resources to train it properly. Combining machine learning with AI and cognitive technologies can make it even more effective in processing large volumes of information.

### 2.2 Information Retrieval

An information retrieval process begins when a user enters a query into the system. Queries are formal statements of information needs, for example search strings in web search engines. In information retrieval a query does not uniquely identify a single object in the collection. Instead, several objects may match the query, perhaps with different degrees of relevancy.

An object is an entity that is represented by information in a content collection or database. User queries are matched against the database information. However, as opposed to classical SQL queries of a database, in information retrieval the results returned may or may not match the query, so results are typically ranked. This ranking of results is a key difference of information retrieval searching compared to database searching.\[5\]

Depending on the application the data objects may be, for example, text documents, images,\[^6\] audio,\[^7\] mind maps\[^8\] or videos. Often the documents themselves are not kept or stored directly in the IR system, but are instead represented in the system by document surrogates or metadata.

Most IR systems compute a numeric score on how well each object in the database matches the query, and rank the objects according to this value. The top ranking objects are then shown to the user. The process may then be iterated if the user wishes to refine the query.

The evaluation of an information retrieval system is the process of assessing how well a system meets the information needs of its users. In general, measure-
CHAPTER 2. BACKGROUND

ment considers a collection of documents to be searched and a search query. Traditional evaluation metrics, designed for Boolean retrieval[clarification needed] or top-k retrieval, include precision and recall. All measures assume a ground truth notion of relevancy: every document is known to be either relevant or non-relevant to a particular query. In practice, queries may be ill-posed and there may be different shades of relevancy.

Relevance Scoring

Lucene (and thus Elasticsearch) uses the Boolean model to find matching documents, and a formula called the practical scoring function to calculate relevance. This formula borrows concepts from term frequency/inverse document frequency and the vector space model but adds more-modern features like a coordination factor, field length normalization, and term or query clause boosting.

Boolean Model

The Boolean model simply applies the AND, OR, and NOT conditions expressed in the query to find all the documents that match. A query for full AND text AND search AND (elasticsearch OR lucene) will include only documents that contain all of the terms full, text, and search, and either elasticsearch or lucene. This process is simple and fast. It is used to exclude any documents that cannot possibly match the query.

Term Frequency/Inverse Document Frequency

Once we have a list of matching documents, they need to be ranked by relevance. Not all documents will contain all the terms, and some terms are more important than others. The relevance score of the whole document depends (in part) on the weight of each query term that appears in that document. The weight of a term is determined by three factors, which we already introduced in What Is Relevance?. The formulae are included for interest’s sake, but you are not required to remember them.

Term Frequency

How often does the term appear in this document? The more often, the higher the weight. A field containing five mentions of the same term is more likely to be relevant than a field containing just one mention. The term frequency is calculated as follows:

$$tf(t \text{ in } d) = \sqrt{frequency}$$

The term frequency (tf) for term t in document d is the square root of the number of times the term appears in the document.
CHAPTER 2. BACKGROUND

Inverse document frequency

How often does the term appear in all documents in the collection? The more often, the lower the weight. Common terms like and or the contribute little to relevance, as they appear in most documents, while uncommon terms like elastic or hippopotamus help us zoom in on the most interesting documents. The inverse document frequency is calculated as follows:

\[ \text{idf}(t) = 1 + \log \left( \frac{\text{numDocs}}{\text{docFreq} + 1} \right) \]

The inverse document frequency (idf) of term \( t \) is the logarithm of the number of documents in the index, divided by the number of documents that contain the term.

Field-length norm

How long is the field? The shorter the field, the higher the weight. If a term appears in a short field, such as a title field, it is more likely that the content of that field is about the term than if the same term appears in a much bigger body field. The field length norm is calculated as follows:

\[ \text{norm}(d) = \frac{1}{\sqrt{\text{frequency}}} \]

The field-length norm (norm) is the inverse square root of the number of terms in the field.

While the field-length norm is important for full-text search, many other fields don’t need norms. Norms consume approximately 1 byte per string field per document in the index, whether or not a document contains the field. Exact-value "not_analyzed" string fields have norms disabled by default, but you can use the field mapping to disable norms on analyzed fields as well: For use cases such as logging, norms are not useful. All you care about is whether a field contains a particular error code or a particular browser identifier. The length of the field does not affect the outcome. Disabling norms can save a significant amount of memory.

Putting it together

These three factors—term frequency, inverse document frequency, and field-length norm—are calculated and stored at index time. Together, they are used to calculate the weight of a single term in a particular document.

The (abbreviated) explanation from the preceding request is as follows:
CHAPTER 2. BACKGROUND

2.3 Natural Language Processing

Natural language processing (NLP) is a subfield of linguistics, computer science, information engineering, and artificial intelligence concerned with the interactions between computers and human (natural) languages, in particular how to program computers to process and analyze large amounts of natural language data. The history of natural language processing (NLP) generally started in the 1950s, although work can be found from earlier periods. In 1950, Alan Turing published an article titled "Computing Machinery and Intelligence" which proposed what is now called the Turing test as a criterion of intelligence. Up to the 1980s, most natural language processing systems were based on complex sets of hand-written rules. Starting in the late 1980s, however, there was a revolution in natural language processing with the introduction of machine learning algorithms for language processing. This was due to both the steady increase in computational power (see Moore’s law) and the gradual lessening of the dominance of Chomskyan theories of linguistics (e.g. transformational grammar), whose theoretical underpinnings discouraged the sort of corpus linguistics that underlies the machine-learning approach to language processing.

I personally split NLP into two main parts. The first being the “low” level procedures which include lexical analysis filtering of unwanted words, syntactical analysis and the “high” level procedure which include the context and semantic analysis which takes place using Machine learning algorithms. The second part also is the field that enjoys the most interest and research in the recent years. We will start by first showcasing some of the “lower” level procedures which also appear in the methodology implementation of the project itself and then continue to the “higher” level, state of the art, Machine Learning Procedures.

Lowercase

An easy to implement but very imperative prepossessing is the normalization of text to lower case. Wherever that is applicable as there are case where Capital Letter give valuable information.
Stopwords

In computing, stop words are words which are filtered out before or after processing of natural language data (text). Though “stop words” usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search. Any group of words can be chosen as the stop words for a given purpose. For some search engines, these are some of the most common, short function words, such as the, is, at, which, and on. In this case, stop words can cause problems when searching for phrases that include them, particularly in names such as “The Who”, “The The”, or “Take That”. Other search engines remove some of the most common words—including lexical words, such as ”want”—from a query in order to improve performance.

Performance, Precision Trade-off

Back in the early days of information retrieval, disk space and memory were limited to a tiny fraction of what we are accustomed to today. It was essential to make your index as small as possible. Every kilobyte saved meant a significant improvement in performance. Stemming (see Reducing Words to Their Root Form) was important, not just for making searches broader and increasing retrieval in the same way that we use it today, but also as a tool for compressing index size. Another way to reduce index size is simply to index fewer words. For search purposes, some words are more important than others. A significant reduction in index size can be achieved by indexing only the more important terms.

So which terms can be left out? We can divide terms roughly into two groups:

• Low-frequency terms
  – Words that appear in relatively few documents in the collection. Because of their rarity, they have a high value, or weight.

• High-frequency terms
  – Common words that appear in many documents in the index, such as the, and, and is. These words have a low weight and contribute little to the relevance score.

Which terms are low or high frequency depend on the documents themselves. The word and may be a low-frequency term if all the documents are in Chinese. In a collection of documents about databases, the word database may be a high-frequency term with little value as a search term for that particular collection.

That said, for any language there are words that occur very commonly and that seldom add value to a search. The default English stopwords used in Elasticsearch are as follows:
These stopwords can usually be filtered out before indexing with little negative impact on retrieval.

Excluding the preceding 33 common words from the index will save only about 4MB per million documents. Using stopwords for the sake of reducing index size is no longer a valid reason. (However, there is one caveat to this statement, which we discuss in Stopwords and Phrase Queries.) On top of that, by removing words from the index, we are reducing our ability to perform certain types of searches. Filtering out the words listed previously prevents us from doing the following:

- Distinguishing happy from not happy.
- Searching for the band The The.
- Finding Shakespeare’s quotation “To be, or not to be”
- Using the country code for Norway: no

The primary advantage of removing stopwords is performance. Imagine that we search an index with one million documents for the word fox. Perhaps fox appears in only 20 of them, which means that Elasticsearch has to calculate the relevance score for 20 documents in order to return the top 10. Now, we change that to a search for the OR fox. The word the probably occurs in almost all the documents, which means that Elasticsearch has to calculate the _score for all one million documents. This second query simply cannot perform as well as the first. Fortunately, there are techniques that we can use to keep common words searchable, while still maintaining good performance.

Reducing to Root Form

This is the step where words are being reduced to their root form. Most languages of the world are inflected, meaning that words can change their form to express difference in the following:

- Number: fox, foxes
- Tense: pay, paid, paying
- Gender: waiter, waitress
- Person: hear, hears
- Case: I, me, my
• Aspect: ate, eaten

• Mood: so be it, were it so

While inflection aids expressivity, it interferes with retrievability, as a single root word sense (or meaning) may be represented by many different sequences of letters. English is a weakly inflected language (you could ignore inflections and still get reasonable search results), but some other languages are highly inflected and need extra work to achieve high-quality search results. Stemming attempts to remove the differences between inflected forms of a word, in order to reduce each word to its root form. For instance, foxes may be reduced to the root fox, to remove the difference between singular and plural in the same way that we removed the difference between lowercase and uppercase.

The root form of a word may not even be a real word. The words jumping and jumpiness may both be stemmed to jumpi. It doesn’t matter—if the same terms are produced at index time and at search time, search will just work. If stemming were easy, there would be only one implementation. Unfortunately, stemming is an inexact science that suffers from two issues: understemming and overstemming. Understemming is the failure to reduce words with the same meaning to the same root. For example, jumped and jumps may be reduced to jump, while jumping may be reduced to jumpi. Understemming reduces retrieval; relevant documents are not returned. Overstemming is the failure to keep two words with distinct meanings separate. For instance, general and generate may both be stemmed to gener. Overstemming reduces precision: irrelevant documents are returned when they shouldn’t be.

**Lemmatization**

A lemma is the canonical, or dictionary, form of a set of related words—the lemma of paying, paid, and pays is pay. Usually the lemma resembles the words it is related to but sometimes it doesn’t— the lemma of is, was, am, and being is be. Lemmatization, like stemming, tries to group related words, but it goes one step further than stemming in that it tries to group words by their word sense, or meaning. The same word may represent two meanings—for example, wake can mean to wake up or a funeral. While lemmatization would try to distinguish these two-word senses, stemming would incorrectly conflate them. Lemmatization is a much more complicated and expensive process that needs to understand the context in which words appear in order to make decisions about what they mean. In practice, stemming appears to be just as effective as lemmatization, but with a much lower cost. In ElasticSearch there are two classes of Stemmers available:

• Algorithmic Stemmers

• Dictionary Stemmers
Algorithmic Stemmers

Most of the stemmers available in Elasticsearch are algorithmic in that they apply a series of rules to a word in order to reduce it to its root form, such as stripping the final s or es from plurals. They don’t have to know anything about individual words in order to stem them. These algorithmic stemmers have the advantage that they are available out of the box, are fast, use little memory, and work well for regular words. The downside is that they don’t cope well with irregular words like be, are, and am, or mice and mouse. One of the earliest stemming algorithms is the Porter stemmer for English, which is still the recommended English stemmer today. Martin Porter subsequently went on to create the “Snowball” (Martin Porter et al.) language for creating stemming algorithms, and a number of the stemmers available in Elasticsearch are written in Snowball. Dictionary stemmers work quite differently from algorithmic stemmers. Instead of applying a standard set of rules to each word, they simply look up the word in the dictionary. Theoretically, they could produce much better results than an algorithmic stemmer. A dictionary stemmer should be able to do the following:

Dictionary Stemmers

Return the correct root word for irregular forms such as feet and mice Recognize the distinction between words that are similar but have different word senses—for example, organ and organization In practice, a good algorithmic stemmer usually outperforms a dictionary stemmer. There are a couple of reasons this should be so:

Dictionary quality

A dictionary stemmer is only as good as its dictionary. The Oxford English Dictionary website estimates that the English language contains approximately 750,000 words (when inflections are included). Most English dictionaries available for computers contain about 10% of those. The meaning of words changes with time. While stemming mobility to mobil may have made sense previously, it now conflates the idea of mobility with a mobile phone. Dictionaries need to be kept current, which is a time-consuming task. Often, by the time a dictionary has been made available, some of its entries are already out-of-date. If a dictionary stemmer encounters a word not in its dictionary, it doesn’t know how to deal with it. An algorithmic stemmer, on the other hand, will apply the same rules as before, correctly or incorrectly.

Size and performance

A dictionary stemmer needs to load all words, all prefixes, and all suffixes into memory. This can use a significant amount of RAM. Finding the right stem for a word is often considerably more complex than the equivalent process with an algorithmic stemmer. Depending on the quality of the dictionary, the process of
removing prefixes and suffixes may be more or less efficient. Less-efficient forms can slow the stemming process significantly. Algorithmic stemmers, on the other hand, are usually simple, small, and fast. In Elasticsearch Dictionary Stemmers are implemented with hunspell. Hunspell hunspell.github.io is the spell checker used by Open Office, LibreOffice, Chrome, Firefox, Thunderbird, and many other open and closed source projects.

**Synonyms**

While stemming helps to broaden the scope of search by simplifying inflected words to their root form, synonyms broaden the scope by relating concepts and ideas. Perhaps no documents match a query for “English queen,” but documents that contain “British monarch” would probably be considered a good match. A user might search for “the US” and expect to find documents that contain United States, USA, U.S.A., America, or the States. However, they wouldn’t expect to see results about the states of matter or state machines. This example provides a valuable lesson. It demonstrates how simple it is for a human to distinguish between separate concepts, and how tricky it can be for mere machines. The natural tendency is to try to provide synonyms for every word in the language, to ensure that any document is findable with even the most remotely related terms. This is a mistake. In the same way that we prefer light or minimal stemming to aggressive stemming, synonyms should be used only where necessary. Users understand why their results are limited to the words in their search query. They are less understanding when their results seems almost random. Synonyms can be used to conflate words that have pretty much the same meaning, such as jump, leap, and hop, or pamphlet, leaflet, and brochure. Alternatively, they can be used to make a word more generic. For instance, bird could be used as a more general synonym for owl or pigeon, and adult could be used for man or woman. Synonyms appear to be a simple concept, but they are quite tricky to get right. In this chapter, we explain the mechanics of using synonyms and discuss the limitations and gotchas. Synonyms are used to broaden the scope of what is considered a matching document. Just as with stemming or partial matching, synonym fields should not be used alone but should be combined with a query on a main field that contains the original text in unadulterated form. See Most Fields for an explanation of how to maintain relevance when using synonyms.

### 2.4 Metrics

**Precision**

Precision is the fraction of the documents retrieved that are relevant to the user’s information need.

\[
\text{precision} = \frac{\text{relevant documents} \cap \text{retrieved documents}}{\text{retrieved documents}}
\]
In binary classification, precision is analogous to positive predictive value. Precision takes all retrieved documents into account. It can also be evaluated at a given cut-off rank, considering only the topmost results returned by the system. This measure is called precision at n or P@n.

Note that the meaning and usage of "precision" in the field of information retrieval differs from the definition of accuracy and precision within other branches of science and statistics.

Recall
Recall is the fraction of the documents that are relevant to the query that are successfully retrieved.

\[
\text{recall} = \frac{\text{non-relevant documents} \cap \text{retrieved documents}}{\text{non-retrieved documents}}
\]

In binary classification, recall is often called sensitivity. So it can be looked at as the probability that a relevant document is retrieved by the query.

It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore, recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by computing the precision.

Mean Precision
Precision and recall are single-value metrics based on the whole list of documents returned by the system. For systems that return a ranked sequence of documents, it is desirable to also consider the order in which the returned documents are presented. By computing a precision and recall at every position in the ranked sequence of documents, one can plot a precision-recall curve, plotting precision \( p(r) \) as a function of recall \( r \). Average precision computes the average value of \( p(r) \) over the interval from \( r = 0 \) to \( r = 1 \)

\[
\text{Avep} = \int_0^1 p(r) dr
\]

That is the area under the precision-recall curve. This integral is in practice replaced with a finite sum over every position in the ranked sequence of documents:

\[
\text{Avep} = \sum_{k=1}^{n} \left( \frac{P(k) \times \text{rel}(k)}{\text{number of relevant documents}} \right)
\]

F-Measure
The weighted harmonic mean of precision and recall, the traditional F-measure or balanced F-score is:

\[
F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{(\text{precision} + \text{recall})}
\]
CHAPTER 2. BACKGROUND

This is also known as the $F_1$ measure, because recall and precision are evenly weighted.

**GMAP**

Mean average precision for a set of queries is the mean of the average precision scores for each query.

$$\text{MAP} = \frac{\sum_{q=1}^{Q} \text{AveP}(q)}{Q}$$

where $Q$ is the number of queries.

**GMAP**

The Geometric Mean Average Precision (GMAP) is the geometric mean of the average precision values for an information retrieval system over a set of $n$ query topics. GMAP is expressed as follows

$$\text{GMAP} = \sqrt[n]{\prod_{i=1}^{n} \text{AP}_n}$$

**Discounted Cumulative Gain**

Discounted Cumulative Gain (DCG) measures the usefulness, or gain, of a document based on its position in the result list. The gain is accumulated from the top of the result list to the bottom, with the gain of each result discounted at lower ranks. Also this is the metric we use in order to do comparison between the different methodologies in sentence embedding techniques.

- Measure of ranking quality.
- Used to measure effectiveness of search algorithms in information retrieval.
- Underlying Assumptions
  - Highly relevant documents are more useful if appearing earlier in search result.
  - Highly relevant documents are more useful than marginally relevant documents which are better than non-relevant documents.
- DCG accumulated at a particular rank position $p$ is given by:

$$\text{DCG}_p = \sum_{i=1}^{p} \frac{\text{rel}_i}{\log_2(i+1)} = \text{rel}_1 + \sum_{i=2}^{p} \frac{\text{rel}_i}{\log_2(i+1)}$$
• Alternative formulation of DCG that places stronger emphasis on retrieving relevant documents is given by:

\[
DCG_p = \sum_{i=1}^{p} \frac{2^{rel_i} - 1}{\log_2(i + 1)}
\]

• Both the alternatives are same if relevance values are binary i.e. \( rel_i \in \{0, 1\} \)

• Various formulations use \( \log_e \) instead of \( \log_2 \)

• Logarithmic scale for reduction provides a smooth reduction curve and hence is used.

• DCG is a successor of Cumulative Gain.

**Cumulative Gain**

• Does not include the position of a result in the calculation of gain of the result set.

• CG at a particular rank position \( p \) is given by:

\[
CG_p = \sum_{i=1}^{p} rel_i
\]

• Where \( rel_i \) is the graded relevance of result at position \( i \).

• So, CG is unaffected by changes in ordering of search results and hence, DCG is used for more accurate measure.

### 2.5 Context

Understanding context is the biggest issue in the in the Question Answering domain. That is mainly because context is also a philosophical issue and has been a case of study for us humans as well. We have yet to define and understand what context is and how it takes its various forms. Also context changes in time. Scharfstein states that 'the problem of context is too difficult for philosophers or anyone else to solve' (1989).

Hermeneutics, interpretation or the art of interpretation, emerged as an important perspective during the Reformation which demanded precise procedures for the accurate interpretation of the Bible. a philosophical tradition that runs from the Protestant theologian Schleiermacher through Dilthey to its more contemporary exponents Heidegger, Dadamer and Ricoeur, hermeneutics. Palmer talks of the 'double focus of hermeneutics', on the theory of understanding in a general sense on the one hand, and on the hermeneutical exegesis of linguistic texts on the other.
CHAPTER 2. BACKGROUND

One issue that has remained central to the hermeneutic project is the concept of the ‘hermeneutic circle’ a relation of part to whole. Palmer gives the following definition:

*We understand the meaning of an individual word by seeing it in reference to the whole sentence; and reciprocally, the sentence’s meaning as a whole is dependent on the meaning of the individual words.*

He then broadens its scope:

*By extension, an individual concept derives its meaning from the context or horizon within which stands; yet the horizon is made up of the very elements to which it gives meaning.*

Scharfstein though doesn’t state anything about algorithms and Machine Learning.

Typically, in Natural Language Processing (NLP), words are treated as the sole data input in each individual processing request. NLP tries to extract the topics, intents, entities (names, account numbers, etc.), and sentiments from this text. But often, there is not enough information to fully express the complete meaning of the sentence. This is because meaning is wrapped up in the context of the conversation. By just looking at words alone, the meaning is often lost, ambiguities persist, and errors are made. It all started with Word2Vec Google on efficient vector representations of words in 2013 by Mikolov et al [11].

But to be able to handle context in Machine Learning we should be able to express it in a more mathematical way. That would be a representation of a sentence with a vector which derives from first representing a word into the form of a vector and then combining those to form the vector for the sentence. This procedure will take into account various facts some of which may be how common is the occurrence of two words in the same sentence and how close or far they are from each other. Thus two vectors that are similar (direction, length, position) will be likely to have similar context or maybe if the directions are opposite show us that they are used in similar context with opposite meanings. These are just a scratch of the effort that has been done in order to express context. a main effort is done in the manner of word embedding.

### 2.6 Evolution of Word Embedding

Even though a part of the NLP Process it deserves to be discussed alone. As its a vast area and one that is being highly developed by the worldwide research community including Facebook, Google, Stanford and others. But State of the art word embedding wasn’t realised in one day.

Computers are unable to understand the concepts of words. In order to process natural language a mechanism for representing text is required. The standard mechanism for text representation are word vectors where words or phrases from a given language vocabulary are mapped to vectors of real numbers. Let us discover the evolution of this research field.
CHAPTER 2. BACKGROUND

2.6.1 Bag of Words (BoW)

BoW vector representations are the most common used traditional vector representation. Each word or n-gram is linked to a vector index and marked as 0 or 1 depending on whether it occurs in a given document.

BoW representations are often used in methods of document classification where the frequency of each word, bi-word or tri-word is a useful feature for training classifiers. One challenge with bag of word representations is that they don’t encode any information with regards to the meaning of a given word.

In BoW word occurrences are evenly weighted independently of how frequently or what context they occur. However in most NLP tasks some words are more relevant than others as we have already seen.

Figure 2.1: An example of a one hot bag of words representation for documents with one word.

| banana | 0 0 0 0 0 0 1 0 0 0 0 0 0 |
| mango  | 0 0 0 0 0 0 0 0 0 0 1 0 0 |

2.6.2 Distributional Embeddings

These enable word vectors to encapsulate contextual context. Each embedding vector is represented based on the mutual information it has with other words in a given corpus. Mutual information can be represented as a global co-occurrence frequency or restricted to a given window either sequentially or based on dependency edges.

2.6.3 Word Embeddings

Sparse vector representations of text, the so-called bag-of-words model have a long history in NLP. Dense vector representations of words or word embeddings have been used as early as 2001 as we have seen above. The main innovation that was proposed in 2013 by Mikolov et al. \[1\] was to make the training of these word embeddings more efficient by removing the hidden layer and approximating the objective. While these changes were simple in nature, they enabled—together with the efficient word2vec implementation—large-scale training of word embeddings.

Word2vec comes in two flavours that can be seen in \[2.3\] below: continuous bag-of-words (CBOW) and skip-gram. They differ in their objective: one predicts the centre word based on the surrounding words, while the other does the opposite.

While these embeddings are no different conceptually than the ones learned with a feed-forward neural network, training on a very large corpus enables them
Figure 2.2: An example distributional embedding matrix each row encodes distributional context based on the count of the words it co-occurs with.

Figure 2.3: Continuous bag-of-words and skip-gram architectures to capture certain relation between words such as gender, verb tense, and country-capital relations, which can be seen in 2.4.

These relations and the meaning behind them sparked initial interest in word embeddings and many studies have investigated the origin of these linear relationships (Arora et al., 2016; Mimno Thompson, 2017; Antoniak Mimno, 2018; Wendlandt et al., 2018) [21] [22] [23] [24]. What cemented word embeddings as a mainstay in current NLP, however, was that using pretrained embeddings as initialization was shown to improve performance across a wide range of downstream tasks [25].
CHAPTER 2. BACKGROUND

2.4 Relations captured by word2vec

Figure 2.4: Relations captured by word2vec

While the relations word2vec captured had an intuitive and almost magical quality to them, later studies showed that there is nothing inherently special about word2vec: Word embeddings can also be learned via matrix factorization (Pennington et al., 2014; Levy Goldberg, 2014) and with proper tuning, classic matrix factorization approaches like SVD and LSA achieve similar results (Levy et al., 2015).

Since then, a lot of work has gone into exploring different facets of word embeddings (as indicated by the staggering number of citations of the original paper). Despite many developments, word2vec is still a popular choice and widely used today. Word2vec’s reach has even extended beyond the word level: skip-gram with negative sampling, a convenient objective for learning embeddings based on local context, has been applied to learn representations for sentences (Mikolov Le, 2014; Kiros et al., 2015)—and even going beyond NLP—to networks (Grover Leskovec, 2016) and biological sequences (Asgari Mofrad, 2015), among others.

One particularly exciting direction is to project word embeddings of different languages into the same space to enable (zero-shot) cross-lingual transfer. It is becoming increasingly possible to learn a good projection in a completely unsupervised way (at least for similar languages), which opens applications for low-resource languages and unsupervised machine translation (Lample et al., 2018; Artetxe et al., 2018). Have a look at (Ruder et al., 2018) for an overview.

2.6.4 Neural Networks for NLP

2013 and 2014 marked the time when neural network models started to get adopted in NLP. Three main types of neural networks became the most widely used: recurrent neural networks, convolutional neural networks, and recursive neural networks.

Recurrent neural networks (RNNs) are an obvious choice to deal with the dynamic input sequences ubiquitous in NLP. Vanilla RNNs (Elman, 1990) were quickly replaced with the classic long-short term memory networks (Hochreiter Schmidhuber, 1997), which proved more resilient to the vanishing and exploding gradient problem. Before 2013, RNNs were still thought to be difficult to
CHAPTER 2. BACKGROUND

train; Ilya Sutskever’s PhD thesis was a key example on the way to changing this reputation. A visualization of an LSTM cell can be seen in 2.5 below. A bidirectional LSTM (Graves et al., 2013) is typically used to deal with both left and right context.

![Figure 2.5: An LSTM network](image)

With convolutional neural networks (CNNs) being widely used in computer vision, they also started to get applied to language (Kalchbrenner et al., 2014; Kim et al., 2014). A convolutional neural network for text only operates in two dimensions, with the filters only needing to be moved along the temporal dimension. 2.6 below shows a typical CNN as used in NLP.

![Figure 2.6: A convolutional neural network for text](image)

An advantage of convolutional neural networks is that they are more parallelizable than RNNs, as the state at every timestep only depends on the local context (via the convolution operation) rather than all past states as in the RNN. CNNs can be extended with wider receptive fields using dilated convolutions to capture a wider context (Kalchbrenner et al., 2016). CNNs and LSTMs can also be combined and stacked and convolutions can be used to speed up an LSTM.

RNNs and CNNs both treat the language as a sequence. From a linguistic perspective, however, language is inherently hierarchical: Words are composed into higher-order phrases and clauses, which can themselves be recursively combined according to a set of production rules. The linguistically inspired idea of treating sentences as trees rather than as a sequence gives rise to recursive neural networks, which can be seen in 2.7 below.
Recursive neural networks build the representation of a sequence from the bottom up in contrast to RNNs who process the sentence left-to-right or right-to-left. At every node of the tree, a new representation is computed by composing the representations of the child nodes. As a tree can also be seen as imposing a different processing order on an RNN, LSTMs have naturally been extended to trees [48].

Not only RNNs and LSTMs can be extended to work with hierarchical structures. Word embeddings can be learned based not only on local but on grammatical context (Levy Goldberg, 2014) [49]; language models can generate words based on a syntactic stack (Dyer et al., 2016) [50]; and graph-convolutional neural networks can operate over a tree (Bastings et al., 2017) [51].

2.6.5 Sequence to sequence models

In 2014, Sutskever et al. [52] proposed sequence-to-sequence learning, a general framework for mapping one sequence to another one using a neural network. In the framework, an encoder neural network process a sentence symbol by symbol and compresses it into a vector representation; a decoder neural network then predicts the output symbol by symbol based on the encoder state, taking as input at every step the previously predicted symbol as can be seen in figure 2.8 below.

Machine translation turned out to be the killer application of this framework. In 2016, Google announced that it was starting to replace its monolithic phrase-based MT models with neural MT models (Wu et al., 2016) [53]. According to Jeff Dean, this meant replacing 500,000 lines of phrase-based MT code with a 500-line neural network model.
CHAPTER 2. BACKGROUND

This framework due to its flexibility is now the go-to framework for natural language generation tasks, with different models taking on the role of the encoder and the decoder. Importantly, the decoder model can not only be conditioned on a sequence, but on arbitrary representations. This enables for instance generating a caption based on an image (Vinyals et al., 2015) \[54\] (as can be seen in 2.9 below), text based on a table (Lebret et al., 2016) \[55\], and a description based on source code changes (Loyola et al., 2017) \[56\], among many other applications.

Sequence-to-sequence learning can even be applied to structured prediction tasks common in NLP where the output has a particular structure. For simplicity, the output is linearized as can be seen for constituency parsing in 2.10 below. Neural networks have demonstrated the ability to directly learn to produce such a linearized output given sufficient amount of training data for constituency parsing (Vinyals et al, 2015) \[57\], and named entity recognition (Gillick et al., 2016) \[58\], among others.

Encoders for sequences and decoders are typically based on RNNs but other model types can be used. New architectures mainly emerge from work in MT,
which acts as a Petri dish for sequence-to-sequence architectures. Recent models are deep LSTMs (Wu et al., 2016) \cite{59}, convolutional encoders (Kalchbrenner et al., 2016; Gehring et al., 2017) \cite{60} \cite{61}, the Transformer (Vaswani et al., 2017) \cite{62}, which will be discussed in the next section, and a combination of an LSTM and a Transformer (Chen et al., 2018) \cite{63}.

### 2.6.6 Attention

Attention (Bahdanau et al., 2015) \cite{64} is one of the core innovations in neural MT (NMT) and the key idea that enabled NMT models to outperform classic phrase-based MT systems. The main bottleneck of sequence-to-sequence learning is that it requires to compress the entire content of the source sequence into a fixed-size vector. Attention alleviates this by allowing the decoder to look back at the source sequence hidden states, which are then provided as a weighted average as additional input to the decoder as can be seen in Figure 2.11 below.

Different forms of attention are available (Luong et al., 2015) \cite{65}. Attention is widely applicable and potentially useful for any task that requires making decisions based on certain parts of the input. It has been applied to constituency parsing (Vinyals et al., 2015) \cite{66}, reading comprehension (Hermann et al., 2015) \cite{67}, and one-shot learning (Vinyals et al., 2016) \cite{68}, among many others. The input does not even need to be a sequence, but can consist of other representations as in the case of image captioning (Xu et al., 2015) \cite{69}, which can be seen in Figure 2.12 below. A useful side-effect of attention is that it provides a rare—if only superficial—glimpse into the inner workings of the model by inspecting which parts of the input are relevant for a particular output based on the attention weights.

Attention is also not restricted to just looking at the input sequence; self-attention can be used to look at the surrounding words in a sentence or document to obtain more contextually sensitive word representations. Multiple layers of self-attention are at the core of the Transformer architecture (Vaswani et al., 2017) \cite{70}, the current state-of-the-art model for NMT.

### 2.6.7 Memory Based Networks

Attention can be seen as a form of fuzzy memory where the memory consists of the past hidden states of the model, with the model choosing what to retrieve
Many models with a more explicit memory have been proposed. They come in different variants such as Neural Turing Machines (Graves et al., 2014) [71], Memory Networks (Weston et al., 2015) [72] and End-to-end Memory Networks (Sukhbaatar et al., 2015) [73], Dynamic Memory Networks (Kumar et al., 2015) [74], the Neural Differentiable Computer (Graves et al., 2016) [75], and the Recurrent Entity Network (Henaff et al., 2017) [76].

Memory is often accessed based on similarity to the current state similar to attention and can typically be written to and read from. Models differ in how they implement and leverage the memory. For instance, End-to-end Memory Networks process the input multiple times and update the memory to enable multiple steps of inference. Neural Turing Machines also have a location-based addressing, which allows them to learn simple computer programs like sorting. Memory-based models are typically applied to tasks, where retaining information over longer time spans should be useful such as language modelling and reading comprehension. The con-
cept of a memory is very versatile: A knowledge base or table can function as a memory, while a memory can also be populated based on the entire input or particular parts of it.

### 2.6.8 Pretrained Language Models

Pretrained word embeddings are context-agnostic and only used to initialize the first layer in our models. In recent months, a range of supervised tasks has been used to pretrain neural networks (Conneau et al., 2017; McCann et al., 2017; Subramanian et al., 2018) \[77\] \[78\] \[79\]. In contrast, language models only require unlabelled text; training can thus scale to billions of tokens, new domains, and new languages. Pretrained language models were first proposed in 2015 (Dai, Le, 2015) \[80\]; only recently were they shown to be beneficial across a diverse range of tasks. Language model embeddings can be used as features in a target model (Peters et al., 2018) \[81\] or a language model can be fine-tuned on target task data (Ramachandran et al., 2017; Howard, Ruder, 2018) \[82\] \[83\]. Adding language model embeddings gives a large improvement over the state-of-the-art across many different tasks as can be seen in 2.13 below.

Pretrained language models have been shown enable learning with significantly less data. As language models only require unlabelled data, they are particularly beneficial for low-resource languages where labelled data is scarce.

**Infersent**

In this paper, they study the task of learning universal representations of sentences, i.e., a sentence encoder model that is trained on a large corpus and subsequently transferred to other tasks. Hence, they investigate the impact of the sentence encoding architecture on representational transferability, and compare convolutional,
CHAPTER 2. BACKGROUND

Figure 2.13: Improvements with language model embeddings over the state-of-the-art

recurrent and even simpler word composition schemes. This paper studies the effects of training sentence embeddings with supervised data by testing on 12 different transfer tasks. We showed that models learned on NLI can perform better than models trained in unsupervised conditions or on other supervised tasks. By exploring various architectures, we showed that a BiLSTM network with max pooling makes the best current universal sentence encoding methods, outperforming existing approaches like SkipThought vectors.

**Universal Sentence Encoder**

We present models for encoding sentences into embedding vectors that specifically target transfer learning to other NLP tasks. In this paper, we present two models for producing sentence embeddings that demonstrate good transfer to a number of other NLP tasks. We include experiments with varying amounts of transfer task training data to illustrate the relationship between transfer task performance and training set size. We make available two new models for encoding sentences into embedding vectors. One makes use of the transformer [12] architecture, while the other is formulated as a deep averaging network (DAN) [13].

**Transformer**

The transformer based sentence encoding model constructs sentence embeddings using the encoding sub-graph of the transformer architecture [12]. This sub-graph
CHAPTER 2. BACKGROUND

uses attention to compute context aware representations of words in a sentence that take into account both the ordering and identity of all the other words. The context aware word representations are converted to a fixed length sentence encoding vector by computing the element-wise sum of the representations at each word position. The encoder takes as input a lowercased PTB tokenized string and outputs a 512 dimensional vector as the sentence encoding. The encoding model is designed to be as general purpose as possible

Deep Averaging Network

The second encoding model makes use of a deep averaging network (DAN) where input embeddings for words and bi-grams are first averaged together and then passed through a feedforward deep neural network (DNN) to produce sentence embeddings. The DAN encoder is trained similarly to the Transformer based encoder. We make use of multitask learning whereby a single DAN encoder is used to supply sentence embeddings for multiple downstream tasks. The primary advantage of the DAN encoder is that compute time is linear in the length of the input sequence. Similar to our results demonstrate that DANs achieve strong baseline performance on text classification tasks.

ELMo

Our representations differ from traditional word type embeddings in that each token is assigned a representation that is a function of the entire input sentence. We use vectors derived from a bidirectional LSTM that is trained with a coupled language model (LM) objective on a large text corpus. For this reason, we call them ELMo (Embeddings from Language Models) representations. Unlike previous approaches for learning contextualized word vectors, ELMo representations are deep, in the sense that they are a function of all of the internal layers of the biLM. More specifically, we learn a linear combination of the vectors stacked above each input word for each end task, which markedly improves performance over just using the top LSTM layer. We first show that they can be easily added to existing models for six diverse and challenging language understanding problems, including textual entailment, question answering and sentiment analysis. The addition of ELMo representations alone significantly improves the state of the art in every case, including up to 20% relative error reductions.

BERT

BERT, which stands for Bidirectional Encoder Representations from Transformers. Unlike recent language representation models, BERT is designed to pre-train deep bidirectional representations by jointly conditioning on both left and right context in all layers. As a result, the pre-trained BERT representations can be fine-tuned with just one additional output layer to create state-of-the-art models for
a wide range of tasks, such as question answering and language inference, without substantial task-specific architecture modifications. BERT is conceptually simple and empirically powerful. It obtains new state-of-the-art results on eleven natural language processing tasks, including pushing the GLUE benchmark to 80.4% (7.6% absolute improvement), MultiNLI accuracy to 86.7% (5.6% absolute improvement) and the SQuAD v1.1 question answering Test F1 to 93.2 (1.5 absolute improvement), outperforming human performance by 2.0.
Chapter 3

Experimental work

3.1 Overview

First the Pubmed dataset was downloaded. PubMed comprises more than 29 million citations for biomedical literature from MEDLINE. After decompressing the data it occupied 235,1 GB of data on the hard disk. From the BioASQ challenge, which will be discussed later, 17 questions were selected in random. These questions will be used to benchmark the performances of the state of the art semantic analysis Infersent Machine Learning model. For this to happen we need a baseline. Our baseline will be also the first step of the methodology, before applying any ML model and is implemented as follows. The dataset will be indexed in the Elasticsearch on a M.2 Solid State Drive (SSD) to increase performance. The Elasticsearch is configured with a set of text Prepossessing and Scoring steps. Namely text analysis, Query and filter context, the combination of the abstract with the title of each paper regarding their amount of weight during scoring. For the final scoring the original unprocessed paper abstract and title are included as well, thus creating the final weighting schema. Using this baseline IR setup we apply queries for the 17 selected questions in the BioASQ database. After retrieving the results we apply some metrics to evaluate the performance. This concludes the baseline of our methodology and referred to as the fist Part of the methodology. Following this first part the retrieved baseline results are reranked using Infersent ML model and re-evaluated. The experiment concludes with the comparison of the 2 states.

3.2 Data Description

3.2.1 Pubmed Dataset

PubMed Central® (PMC) is a free full-text archive of biomedical and life sciences journal literature at the U.S. National Institutes of Health’s National Library of Medicine (NIH/NLM). In keeping with NLM’s legislative mandate to collect and preserve the biomedical literature, PMC serves as a digital counterpart to NLM’s
extensive print journal collection.

PMC was developed and is managed by NLM’s National Center for Biotechnology Information (NCBI).

Since its inception in 2000, PMC has grown from comprising only two journals, PNAS: Proceedings of the National Academy of Sciences and Molecular Biology of the Cell, to an archive of articles from thousands of journals.

Today, PMC contains more than 5 million full-text records, spanning several centuries of biomedical and life science research (late 1700s to present). Content is added to the archive through the resulting collaborations with publishers, societies, research funders, and international organizations form the foundation of PMC. PMC is not a publisher and does not publish journal articles itself.


PubMed comprises more than 29 million citations for biomedical literature from MEDLINE, life science journals, and online books. Citations may include links to full-text content from PubMed Central and publisher web sites.

3.2.2 BioASQ Challenge

BioASQ organizes challenges on biomedical semantic indexing and question answering (QA). The challenges include tasks relevant to hierarchical text classification, machine learning, information retrieval, QA from texts and structured data, multi-document summarization and many other areas. BioASQ consists of two tasks.

BioASQ Task A: Large-scale online biomedical semantic indexing

This task will be based on the standard process followed by PubMed to index journal abstracts. The participants are asked to classify new PubMed documents, written in English, as they become available online, before PubMed curators annotate (in effect, classify) them manually. The classes will come from the MeSH hierarchy; they will be the subject headings that are currently used to manually index the abstracts, excluding those that are already provided by the authors of each article. As new manual annotations become available, they will be used to evaluate the classification performance of participating systems (that classify articles before they are manually annotated), using standard IR measures (e.g., precision, recall, accuracy), as well as hierarchical variants of them. The participants will be able to train their classifiers, using the whole history of manually annotated abstracts.

BioASQ Task B: Biomedical Semantic QA (involves IR, QA, summarization)

Task B will use benchmark datasets containing training and test biomedical questions, in English, along with gold standard (reference) answers. The participants
CHAPTER 3. EXPERIMENTAL WORK

will have to respond to each test question with relevant concepts (from designated terminologies and ontologies), relevant articles (in English, from designated article repositories), relevant snippets (from the relevant articles), relevant RDF triples (from designated ontologies), exact answers (e.g., named entities in the case of factoid questions) and ‘ideal’ answers (English paragraph-sized summaries). 2747 training questions (that were used as dry-run or test questions in previous year) are already available, along with their gold standard answers (relevant concepts, articles, snippets, exact answers, summaries). At least 500 new test questions will be used this year. All the questions are constructed by biomedical experts from around Europe.

3.3 Bioasq Results

Here we can see the results from the task 6B phase A, test batch 1, of the BioASQ competition

<table>
<thead>
<tr>
<th>Documents</th>
<th>Mean precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>MAP</th>
<th>GMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>aueb-nlp-3</td>
<td>0.3015</td>
<td>0.6387</td>
<td>0.3438</td>
<td>0.2327</td>
<td>0.0686</td>
</tr>
<tr>
<td>aueb-nlp-4</td>
<td>0.2965</td>
<td>0.6220</td>
<td>0.3365</td>
<td>0.2288</td>
<td>0.0574</td>
</tr>
<tr>
<td>aueb-nlp-2</td>
<td>0.2945</td>
<td>0.6307</td>
<td>0.3364</td>
<td>0.2275</td>
<td>0.0777</td>
</tr>
<tr>
<td>aueb-nlp-1</td>
<td>0.2965</td>
<td>0.6260</td>
<td>0.3372</td>
<td>0.2265</td>
<td>0.0639</td>
</tr>
<tr>
<td>sdm/rerank</td>
<td>0.2500</td>
<td>0.5648</td>
<td>0.2886</td>
<td>0.1840</td>
<td>0.0277</td>
</tr>
<tr>
<td>testtext</td>
<td>0.2390</td>
<td>0.4206</td>
<td>0.2618</td>
<td>0.1742</td>
<td>0.0099</td>
</tr>
<tr>
<td>ustb_prir1</td>
<td>0.2390</td>
<td>0.4206</td>
<td>0.2618</td>
<td>0.1742</td>
<td>0.0099</td>
</tr>
<tr>
<td>ustb_prir3</td>
<td>0.2380</td>
<td>0.4193</td>
<td>0.2605</td>
<td>0.1738</td>
<td>0.0115</td>
</tr>
<tr>
<td>ustb_prir4</td>
<td>0.2380</td>
<td>0.4193</td>
<td>0.2605</td>
<td>0.1738</td>
<td>0.0115</td>
</tr>
<tr>
<td>ustb_prir2</td>
<td>0.2310</td>
<td>0.4041</td>
<td>0.2519</td>
<td>0.1709</td>
<td>0.0075</td>
</tr>
<tr>
<td>OAQA based system</td>
<td>0.1385</td>
<td>0.3352</td>
<td>0.1581</td>
<td>0.1064</td>
<td>0.0014</td>
</tr>
<tr>
<td>htersys</td>
<td>0.0030</td>
<td>0.0075</td>
<td>0.0043</td>
<td>0.0030</td>
<td>0.0000</td>
</tr>
<tr>
<td>MindLab QA System ++</td>
<td>0.0030</td>
<td>0.0031</td>
<td>0.0030</td>
<td>0.0016</td>
<td>0.0000</td>
</tr>
<tr>
<td>MindLab QA System</td>
<td>0.0031</td>
<td>0.0031</td>
<td>0.0030</td>
<td>0.0012</td>
<td>0.0000</td>
</tr>
</tbody>
</table>

Table 3.1

3.4 Elasticsearch Analysis

Analysis is the process of converting text, like the body of any email, into tokens or terms which are added to the inverted index for searching. Analysis is performed
CHAPTER 3. EXPERIMENTAL WORK

Table 3.2

<table>
<thead>
<tr>
<th>System</th>
<th>Mean precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>MAP</th>
<th>GMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>aueb-nlp-3</td>
<td>0.2224</td>
<td>0.2593</td>
<td>0.2004</td>
<td>0.1684</td>
<td>0.0118</td>
</tr>
<tr>
<td>aueb-nlp-1</td>
<td>0.2156</td>
<td>0.2496</td>
<td>0.1938</td>
<td>0.1675</td>
<td>0.0108</td>
</tr>
<tr>
<td>aueb-nlp-4</td>
<td>0.2127</td>
<td>0.2501</td>
<td>0.1924</td>
<td>0.1659</td>
<td>0.0111</td>
</tr>
<tr>
<td>aueb-nlp-2</td>
<td>0.2168</td>
<td>0.2599</td>
<td>0.198</td>
<td>0.1642</td>
<td>0.0123</td>
</tr>
<tr>
<td>ustb_prir4</td>
<td>0.1634</td>
<td>0.1619</td>
<td>0.1364</td>
<td>0.1209</td>
<td>0.0015</td>
</tr>
<tr>
<td>ustb_prir2</td>
<td>0.1572</td>
<td>0.1616</td>
<td>0.1319</td>
<td>0.1205</td>
<td>0.0019</td>
</tr>
<tr>
<td>ustb_prir3</td>
<td>0.1691</td>
<td>0.168</td>
<td>0.1393</td>
<td>0.1193</td>
<td>0.0018</td>
</tr>
<tr>
<td>testtext</td>
<td>0.1664</td>
<td>0.1723</td>
<td>0.1409</td>
<td>0.1151</td>
<td>0.0022</td>
</tr>
<tr>
<td>ustb_prir1</td>
<td>0.1664</td>
<td>0.1723</td>
<td>0.1409</td>
<td>0.1151</td>
<td>0.0022</td>
</tr>
<tr>
<td>OAQA based system</td>
<td>0.1318</td>
<td>0.1837</td>
<td>0.1322</td>
<td>0.1089</td>
<td>0.0005</td>
</tr>
<tr>
<td>sdm/rerank</td>
<td>0.094</td>
<td>0.087</td>
<td>0.0722</td>
<td>0.0572</td>
<td>0.0004</td>
</tr>
<tr>
<td>MindLab QA System</td>
<td>0.0014</td>
<td>0.0013</td>
<td>0.0013</td>
<td>0.0004</td>
<td>0</td>
</tr>
<tr>
<td>MindLab QA System ++</td>
<td>0.0014</td>
<td>0.0013</td>
<td>0.0013</td>
<td>0.0004</td>
<td>0</td>
</tr>
<tr>
<td>htersys</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 3.3

<table>
<thead>
<tr>
<th>System</th>
<th>Mean precision</th>
<th>Recall</th>
<th>F-Measure</th>
<th>MAP</th>
<th>GMAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>ustb_prir1</td>
<td>0.2019</td>
<td>0.5896</td>
<td>0.2688</td>
<td>0.1623</td>
<td>0.0117</td>
</tr>
<tr>
<td>ustb_prir4</td>
<td>0.187</td>
<td>0.509</td>
<td>0.2466</td>
<td>0.136</td>
<td>0.0147</td>
</tr>
<tr>
<td>sdm/rerank</td>
<td>0.5313</td>
<td>0.4515</td>
<td>0.436</td>
<td>0.1315</td>
<td>0.0186</td>
</tr>
<tr>
<td>testtext</td>
<td>0.1674</td>
<td>0.5154</td>
<td>0.225</td>
<td>0.1246</td>
<td>0.0087</td>
</tr>
<tr>
<td>ustb_prir2</td>
<td>0.1609</td>
<td>0.5015</td>
<td>0.216</td>
<td>0.1117</td>
<td>0.0094</td>
</tr>
<tr>
<td>ustb_prir3</td>
<td>0.0957</td>
<td>0.3159</td>
<td>0.1405</td>
<td>0.0522</td>
<td>0.0005</td>
</tr>
<tr>
<td>OAQA based system</td>
<td>0.0174</td>
<td>0.0428</td>
<td>0.0245</td>
<td>0.011</td>
<td>0</td>
</tr>
<tr>
<td>htersys</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aueb-nlp-1</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aueb-nlp-2</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aueb-nlp-3</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>aueb-nlp-4</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MindLab QA System</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>MindLab QA System ++</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

by an analyzer which can be either a built-in analyzer or a custom analyzer defined
Anatomy of an analyzer

An analyzer — whether built-in or custom — is just a package which contains three lower-level building blocks: character filters, tokenizers, and token filters.

The built-in analyzers pre-package these building blocks into analyzers suitable for different languages and types of text.

Character filters

A character filter receives the original text as a stream of characters and can transform the stream by adding, removing, or changing characters. For instance, a character filter could be used to convert Hindu-Arabic numerals (١٢٣٤٥٦٧٨٩) into their Arabic-Latin equivalents (0123456789), or to strip HTML elements like <b> from the stream. An analyzer may have zero or more character filters, which are applied in order.

Tokenizer

A tokenizer receives a stream of characters, breaks it up into individual tokens (usually individual words), and outputs a stream of tokens. For instance, a whitespace tokenizer breaks text into tokens whenever it sees any whitespace. It would convert the text "Quick brown fox!" into the terms [Quick, brown, fox!].

The tokenizer is also responsible for recording the order or position of each term and the start and end character offsets of the original word which the term represents.

An analyzer must have exactly one tokenizer.

Token filters

A token filter receives the token stream and may add, remove, or change tokens. For example, a lowercase token filter converts all tokens to lowercase, a stop token filter removes common words (stop words) like the from the token stream, and a synonym token filter introduces synonyms into the token stream.

Token filters are not allowed to change the position or character offsets of each token.

An analyzer may have zero or more token filters, which are applied in order.

Analyzers

Elasticsearch ships with a wide range of built-in analyzers:

• Standard Analyzer
The standard analyzer which is also the analyzer that is being used in this work divides text into terms on word boundaries, as defined by the Unicode Text Segmentation algorithm. It removes most punctuation, lowercases terms, and supports removing stop words.

- **Simple Analyzer**  
The simple analyzer divides text into terms whenever it encounters a character which is not a letter. It lowercases all terms.

- **Whitespace Analyzer**  
The whitespace analyzer divides text into terms whenever it encounters any whitespace character. It does not lowercase terms.

- **Stop Analyzer**  
The stop analyzer is like the simple analyzer, but also supports removal of stop words.

- **Keyword Analyzer**  
The keyword analyzer is a “noop” analyzer that accepts whatever text it is given and outputs the exact same text as a single term.

- **Pattern Analyzer**  
The pattern analyzer uses a regular expression to split the text into terms. It supports lower-casing and stop words.

- **Language Analyzers**  
Elasticsearch provides many language-specific analyzers like english or french.

- **Fingerprint Analyzer**  
The fingerprint analyzer is a specialist analyzer which creates a fingerprint which can be used for duplicate detection.

### 3.5 Query and Filter context

The behaviour of a query clause depends on whether it is used in query context or in filter context:

- **Query context**  
A query clause used in query context answers the question “How well does this document match this query clause?” Besides deciding whether or not the document matches, the query clause also calculates a score representing how well the document matches, relative to other documents.
• **Filter context**
  In filter context, a query clause answers the question “Does this document match this query clause?” The answer is a simple Yes or No — no scores are calculated. Filter context is mostly used for filtering structured data, e.g.
  
  - Does this timestamp fall into the range 2015 to 2016?
  - Is the status field set to ”published”?

### 3.6 Full Text Queries

The high-level full text queries are usually used for running full text queries on full text fields like the body of an email. They understand how the field being queried is analyzed and will apply each field’s analyzer (or search_analyzer) to the query string before executing.

The queries in this group are:

• **match query**
  The standard query for performing full text queries, including fuzzy matching and phrase or proximity queries.

• **match_phrase query**
  Like the match query but used for matching exact phrases or word proximity matches.

• **match_phrase_prefix query**
  The poor man’s search-as-you-type. Like the match_phrase query, but does a wildcard search on the final word.

• **multi_match query**
  The multi-field version of the match query.

• **common terms query**
  A more specialized query which gives more preference to uncommon words.

• **query_string query**
  Supports the compact Lucene query string syntax, allowing you to specify AND|OR|NOT conditions and multi-field search within a single query string. For expert users only.

• **simple_query_string query**
  A simpler, more robust version of the query_string syntax suitable for exposing directly to users.
3.7 Multi Match Query

The multi_match query builds on the match query to allow multi-field queries. Each field may be boosted individually. Thus allowing to boost the Title over the abstract itself.

Types of multi_match query

The way the multi_match query is executed internally depends on the type parameter, which can be set to:

- **best_fields**
  (default) Finds documents which match any field, but uses the _score from the best field.

- **most_fields**
  Finds documents which match any field and combines the _score from each field.

- **cross_fields**
  Treats fields with the same analyzer as though they were one big field. Looks for each word in any field.

- **phrase**
  Runs a match_phrase query on each field and uses the _score from the best field.

- **phrase_prefix**
  Runs a match_phrase_prefix query on each field and combines the _score from each field.

**most_fields**

The most_fields type is most useful when querying multiple fields that contain the same text analyzed in different ways. For instance, the main field may contain synonyms, stemming and terms without diacritics. A second field may contain the original terms, and a third field might contain shingles. By combining scores from all three fields we can match as many documents as possible with the main field, but use the second and third fields to push the most similar results to the top of the list. The score from each match clause is added together, then divided by the number of match clauses.
3.8 Phase One: Common Terms Query

The common terms query is a modern alternative to stopwords which improves the precision and recall of search results (by taking stopwords into account), without sacrificing performance.

The problem

Every term in a query has a cost. A search for "The brown fox" requires three term queries, one for each of "the", "brown" and "fox", all of which are executed against all documents in the index. The query for "the" is likely to match many documents and thus has a much smaller impact on relevance than the other two terms.

Previously, the solution to this problem was to ignore terms with high frequency. By treating "the" as a stopword, we reduce the index size and reduce the number of term queries that need to be executed.

The problem with this approach is that, while stopwords have a small impact on relevance, they are still important. If we remove stopwords, we lose precision, (eg we are unable to distinguish between "happy" and "not happy") and we lose recall (eg text like "The The" or "To be or not to be" would simply not exist in the index).

The solution

The common terms query divides the query terms into two groups: more important (ie low frequency terms) and less important (ie high frequency terms which would previously have been stopwords).

First it searches for documents which match the more important terms. These are the terms which appear in fewer documents and have a greater impact on relevance.

Then, it executes a second query for the less important terms — terms which appear frequently and have a low impact on relevance. But instead of calculating the relevance score for all matching documents, it only calculates the _score for documents already matched by the first query. In this way the high frequency terms can improve the relevance calculation without paying the cost of poor performance.

If a query consists only of high frequency terms, then a single query is executed as an AND (conjunction) query, in other words all terms are required. Even though each individual term will match many documents, the combination of terms narrows down the resultset to only the most relevant. The single query can also be executed as an OR with a specific minimum_should_match, in this case a high enough value should probably be used.

Terms are allocated to the high or low frequency groups based on the cutoff_frequency, which can be specified as an absolute frequency (>=1) or as a relative frequency (0.0 .. 1.0). (Remember that document frequencies are computed on a per shard level as explained in the blog post Relevance is broken.)
Perhaps the most interesting property of this query is that it adapts to domain specific stopwords automatically. For example, on a video hosting site, common terms like "clip" or "video" will automatically behave as stopwords without the need to maintain a manual list.

**Query experimentation**

A query experimentation of some of the aforementioned query types was performed to choose the first step for our final experiment which is to decide a top-k methodology configuration in order to get some desirable results. For the Analysis we used as mentioned before the standard analyzer. Title and abstract fields are both parsed with this analyzer and their results are denoted as Title.std and Abstract.std respectively. Each field may be applied with a weight. This is specified using the ^ after the name of the desired field. The configurations used were the following:

- **Configuration 1**
  A Multi Match Query with a most_fields with the weighting of the different types being the following: [Title ^2, Title.std ^1.5, Abstract ^1.5, Abstract.std]

- **Configuration 2**
  A Multi Match Query with a most_fields with the weighting of the different types being the following: [Title ^8, Title.std ^4, Abstract ^2, Abstract.std]

- **Configuration 3**
  In our last configuration the Common Terms Query was used with a cut off frequency between the common and uncommon words being set at 0.1%

**Experimentation Description**

For every query in the BioASQ Data set (2747 of them) were used to make a query on the elasticsearch indexed dataset of the PubMed Data. Each Query in the BioASQ Dataset has a set of golden documents in which the answer or part of it is included. Thus an ideal top-k would be one that has a high percentage of golden documents included in it for most of the queries. Five k sizes were used in our evaluation [50, 100, 250, 500, 1000, 2000, 5000, 10000], in order to see how size affects our results. For each size the parameters we look at are the amount of queries that had at least one document retrieved in the corresponding top-k retrieved documents. For those documents we get the mean, median, minimum and maximum of the percentage of the golden documents included in the size specific retrieved documents. This is performed for each of the configurations.
Experimentation Results

The results for the configurations 1, 2 and 3 are shown in the tables 3.4, 3.5 and 3.6 respectively.

<table>
<thead>
<tr>
<th>k size</th>
<th>non zero elements</th>
<th>mean</th>
<th>median</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>25</td>
<td>0.08357</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.2500</td>
</tr>
<tr>
<td>100</td>
<td>31</td>
<td>0.08219</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.2500</td>
</tr>
<tr>
<td>250</td>
<td>40</td>
<td>0.09309</td>
<td>0.06905</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>500</td>
<td>45</td>
<td>0.09371</td>
<td>0.06667</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>1000</td>
<td>50</td>
<td>0.09052</td>
<td>0.06275</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>2000</td>
<td>56</td>
<td>0.08908</td>
<td>0.06275</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>5000</td>
<td>67</td>
<td>0.08510</td>
<td>0.06250</td>
<td>0.0109</td>
<td>0.5000</td>
</tr>
<tr>
<td>10000</td>
<td>84</td>
<td>0.08593</td>
<td>0.06250</td>
<td>0.0109</td>
<td>0.5000</td>
</tr>
</tbody>
</table>

Table 3.4: Table with the configuration 1 analyzed for each top k size and retrieved results non zero elements, mean and average and also the minimum non zero element and the maximum one

<table>
<thead>
<tr>
<th>k size</th>
<th>non zero elements</th>
<th>mean</th>
<th>median</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>28</td>
<td>0.08594</td>
<td>0.06275</td>
<td>0.0286</td>
<td>0.2500</td>
</tr>
<tr>
<td>100</td>
<td>36</td>
<td>0.09767</td>
<td>0.07418</td>
<td>0.0286</td>
<td>0.5000</td>
</tr>
<tr>
<td>250</td>
<td>43</td>
<td>0.09584</td>
<td>0.06667</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>500</td>
<td>44</td>
<td>0.09650</td>
<td>0.06905</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>1000</td>
<td>55</td>
<td>0.09054</td>
<td>0.06667</td>
<td>0.0179</td>
<td>0.5000</td>
</tr>
<tr>
<td>2000</td>
<td>63</td>
<td>0.08853</td>
<td>0.07143</td>
<td>0.0109</td>
<td>0.5000</td>
</tr>
<tr>
<td>5000</td>
<td>72</td>
<td>0.08752</td>
<td>0.06458</td>
<td>0.0109</td>
<td>0.5000</td>
</tr>
<tr>
<td>10000</td>
<td>82</td>
<td>0.08997</td>
<td>0.06066</td>
<td>0.0109</td>
<td>0.5000</td>
</tr>
</tbody>
</table>

Table 3.5: Table with the configuration 2 analyzed for each top k size and retrieved results non zero elements, mean and average and also the minimum non zero element and the maximum one

By observing the tables we can see how much inferior is the common terms configuration to the other 2. That may be mainly because only the abstract is in use and also no standard analysing is used. It is inferior in all but one field. The least non-zero percentage of relevant retrieved documents is higher that the other 2 configurations in a consistent way
CHAPTER 3. EXPERIMENTAL WORK

### Table 3.6: Table with the configuration 3 analyzed for each top k size and retrieved results non zero elements, mean and average and also the minimum non zero element and the maximum one

<table>
<thead>
<tr>
<th>k size</th>
<th>non zero elements</th>
<th>mean</th>
<th>median</th>
<th>min</th>
<th>max</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>3</td>
<td>0.05112</td>
<td>0.05882</td>
<td>0.0357</td>
<td>0.0588</td>
</tr>
<tr>
<td>100</td>
<td>6</td>
<td>0.07000</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>250</td>
<td>7</td>
<td>0.06893</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>500</td>
<td>8</td>
<td>0.06512</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>1000</td>
<td>8</td>
<td>0.06512</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>2000</td>
<td>9</td>
<td>0.06318</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>5000</td>
<td>10</td>
<td>0.06400</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
<tr>
<td>10000</td>
<td>11</td>
<td>0.07334</td>
<td>0.05882</td>
<td>0.0286</td>
<td>0.1667</td>
</tr>
</tbody>
</table>

3.9 Phase Two: Reranking using sentence embedding

After phase one where Elasticsearch returns results for each of our query, ranking them based on the configuration chosen as shown previously. We select the top-k to be 1000 top results from Elasticsearch and a minimum of 0.1 percentage of the golden documents to be present in the results. This provides us 17 queries as presented in [3.7] and their Mapping to integers also, so that we can refer to them easily. First we present results from the Baseline which is the results from the Elasticsearch itself.

As presented in [3.8] we will compare our final results based on how many golden documents where there in each top-k. The percentage of the retrieved golden documents taking into account the total golden documents and for which we set the threshold of 0.1. The average position of the retrieved golden documents. Finally the DCG, the reason we don’t use nDCG is that our document relevance is binary and the golden documents have the same importance. Also the final evaluation will be done for each query separately so we wont have a cross-query comparison issue. Unfortunately the retrieved documents are too little for us to have a clear view but even with 1 or 2 documents in a thousand the potentials powers of a word embedding should be apparent.

After the application of InferSent sentence embedding on our common Elasticsearch results and the provided query. We append the title in the abstract and simply use it as a sentence in a paragraph, then embeddings are computed. We apply a cosine similarity between each sentence embedding and the query embedding. We then select from each paragraph (appended title and abstract) the sentence with the maximum (MAX), minimum (MIN) and average (AVG) and their equivalent absolute versions (ABS) similarity scores and re-rank our results based on those three parameters and compare results. Below [3.9] we can see how the reranking
### Queries in Experiment

<table>
<thead>
<tr>
<th>Queries</th>
<th>Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Does thyroid hormone regulate calcium transient in the myocardium?</td>
<td>1</td>
</tr>
<tr>
<td>Which is the most typical peptide sequence responsible for retrieval of endoplasmic reticulum (ER) lumenal proteins from the Golgi apparatus?</td>
<td>2</td>
</tr>
<tr>
<td>Does melanoma occur in people of African origin?</td>
<td>3</td>
</tr>
<tr>
<td>Which is the molecular weight of the protein angiogenin?</td>
<td>4</td>
</tr>
<tr>
<td>What are the most frequent non-canonical sequence motifs at the donor and acceptor splice sites in vertebrates?</td>
<td>5</td>
</tr>
<tr>
<td>What type of arrhythmia is known as bidirectional ventricular tachycardia (BDVT)?</td>
<td>6</td>
</tr>
<tr>
<td>What is the action of molindone?</td>
<td>7</td>
</tr>
<tr>
<td>How are thyroid hormones involved in the development of diabetic cardiomyopathy?</td>
<td>8</td>
</tr>
<tr>
<td>Which is the binding site motif of Sp1?</td>
<td>9</td>
</tr>
<tr>
<td>Which is the major phytoalexin in alfalfa (Medicago sativa L.)?</td>
<td>10</td>
</tr>
<tr>
<td>Do A-type lamins bind euchromatin or heterochromatin?</td>
<td>11</td>
</tr>
<tr>
<td>Is calcium overload involved in the development of diabetic cardiomyopathy?</td>
<td>12</td>
</tr>
<tr>
<td>Which is the largest metabolic gene cluster in yeast?</td>
<td>13</td>
</tr>
<tr>
<td>What fruit causes Jamaican vomiting sickness?</td>
<td>14</td>
</tr>
<tr>
<td>List clinical features of EEM syndrome.</td>
<td>15</td>
</tr>
<tr>
<td>List symptoms of the Zieve’s syndrome.</td>
<td>16</td>
</tr>
<tr>
<td>Which bacteria was EcoRI, restriction endonuclease isolated from?</td>
<td>17</td>
</tr>
</tbody>
</table>

Table 3.7: Queries in usage and their corresponding integer mapping

**performed using the Intersent and ranking results using the maximum cosine similarity of each paragraph.**

We observe that other fields are obviously increased by the word embedding and others that got worse positions that would be mainly because we don’t give any weights to the title, like its done to the elasticsearch results and we use a very simple methods to rank a paragraph and that is with a mere MAX, MIN or AVG while other more accurate methodologies could be used. The rest of the combinations (MIN, AVG, ABS MIN, ABS AVG) were tested but due to pure performance are not presented. Intersent with MAX (same results with ABS MAX) had better results in 8 out of 12 Queries, and following is the vector of differences re-ranking minus baseline:
### Baseline Results

<table>
<thead>
<tr>
<th>Index</th>
<th>documents</th>
<th>percentage in golden</th>
<th>average position</th>
<th>DCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.25</td>
<td>12.0</td>
<td>0.263</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.12</td>
<td>367.0</td>
<td>0.117</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.11</td>
<td>667.5</td>
<td>0.214</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.11</td>
<td>142.0</td>
<td>0.139</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.12</td>
<td>72.0</td>
<td>0.161</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.14</td>
<td>24.0</td>
<td>0.213</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.12</td>
<td>60.0</td>
<td>0.168</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.17</td>
<td>983.0</td>
<td>0.101</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.17</td>
<td>39.0</td>
<td>0.187</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.29</td>
<td>178.5</td>
<td>0.268</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0.12</td>
<td>59.0</td>
<td>0.169</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0.17</td>
<td>11.0</td>
<td>0.270</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0.50</td>
<td>73.0</td>
<td>0.161</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.12</td>
<td>3.0</td>
<td>0.431</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0.11</td>
<td>4.0</td>
<td>0.387</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0.14</td>
<td>13.0</td>
<td>0.532</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0.25</td>
<td>34.0</td>
<td>0.193</td>
</tr>
</tbody>
</table>

Table 3.8: Initial ranked documents as elasticsearch scored them according to the configuration chosen

[0.016, 0.03, 0.183, 0.066, -0.018, -0.072, -0.067, 0.053, -0.083, 0.009, -0.067, -0.082, -0.04, -0.1159, -0.264, 0.128, 0.002]
<table>
<thead>
<tr>
<th>Index</th>
<th>documents</th>
<th>percentage in golden</th>
<th>average position</th>
<th>DCG</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>0.25</td>
<td>10.0</td>
<td>0.279</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>0.12</td>
<td>105.0</td>
<td>0.148</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>0.11</td>
<td>31.5</td>
<td>0.397</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>0.11</td>
<td>27.0</td>
<td>0.206</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>0.12</td>
<td>127.0</td>
<td>0.143</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>0.14</td>
<td>134.0</td>
<td>0.141</td>
</tr>
<tr>
<td>7</td>
<td>1</td>
<td>0.12</td>
<td>961.0</td>
<td>0.101</td>
</tr>
<tr>
<td>8</td>
<td>1</td>
<td>0.17</td>
<td>85.0</td>
<td>0.155</td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>0.17</td>
<td>758.0</td>
<td>0.104</td>
</tr>
<tr>
<td>10</td>
<td>2</td>
<td>0.29</td>
<td>149.0</td>
<td>0.277</td>
</tr>
<tr>
<td>11</td>
<td>1</td>
<td>0.12</td>
<td>881.0</td>
<td>0.102</td>
</tr>
<tr>
<td>12</td>
<td>1</td>
<td>0.17</td>
<td>38.0</td>
<td>0.188</td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>0.50</td>
<td>306.0</td>
<td>0.121</td>
</tr>
<tr>
<td>14</td>
<td>1</td>
<td>0.12</td>
<td>7.0</td>
<td>0.315</td>
</tr>
<tr>
<td>15</td>
<td>1</td>
<td>0.11</td>
<td>280.0</td>
<td>0.123</td>
</tr>
<tr>
<td>16</td>
<td>2</td>
<td>0.14</td>
<td>38.0</td>
<td>0.660</td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>0.25</td>
<td>33.0</td>
<td>0.195</td>
</tr>
</tbody>
</table>

Table 3.9: Results of the elasticsearch reranked by Infersent
Chapter 4

Related Work

The task of re-ranking documents and sentences in order to retrieve better and richer in knowledge answers. By leveraging text that is relevant in context and not by sheer Information Retrieval techniques is a task that is of high priority in the research community. A few examples

[17] Uses a system of stacked Bidirectional Long-Short Term Memory (BLSTM) network and output relevance scores between queries and answers. Does not require any prior syntactic parsing or external knowledge resources.

[18] Both the input and the output projections are taken into account in order to get richer results. The proposed Dual Embedding Space Model (DESM) in addition to the traditional term-frequency, captures the evidence on whether a document is about a query. However for large datasets its prone to false positives.

[19] In this study they propose two neural networks rankers that assign scores to different passages based on their likelihood of containing the answer to a given question. In this study no search engines are used as the QUASAR-T provides 100 short passages already retrieved by a search engine for each question. Two rankers are in question. One Infersent ranker to evaluate the performance of semantic similarity in ranking for QA and a Relation-Networks ranker to evaluate the performance of relevance matching.
Chapter 5

Conclusions

We reviewed most of the methodologies for most of the common query retrieval set a basic baseline and then tried to improve that with state of the art sentence embedding technologies. We specifically used Infersent from Facebook and we showed even with simple configuration and ranking methodologies it mainly performs better at ranking documents. This gives us hope for the future usage of these technologies in the Question Answering field. All the steps in this works pipeline need extensive experimentation and thorough thought in order to achieve a combination that has the best results. Specifically more work for the first phase of the common query retrieval should be done with more configurations. So that we will retrieve more golden documents in the first top-k results. Next regarding phase two and sentence embedding with Infersent more combinations of the embeddings and different weights on titles and abstracts, on par with the combination of document ranking methodology (i.e. MIN, MAX, AVG) using the cosine similarity or even other type of similarity should be done. Furthermore there are other sentence embedding techniques like Universal Encoder, ELMo and BERT might be found to be more suitable. Last but not least the vocabulary that was used to train the sentence embeddings was a generic fastText with 2 million word vectors trained on Common Crawl. So probably a domain specific vocabulary would make a vary good improvement on our results. Since such research is not found in the bio-medicine domain it could also be our next target to publish a paper on that issue.
Bibliography


[20] temp


